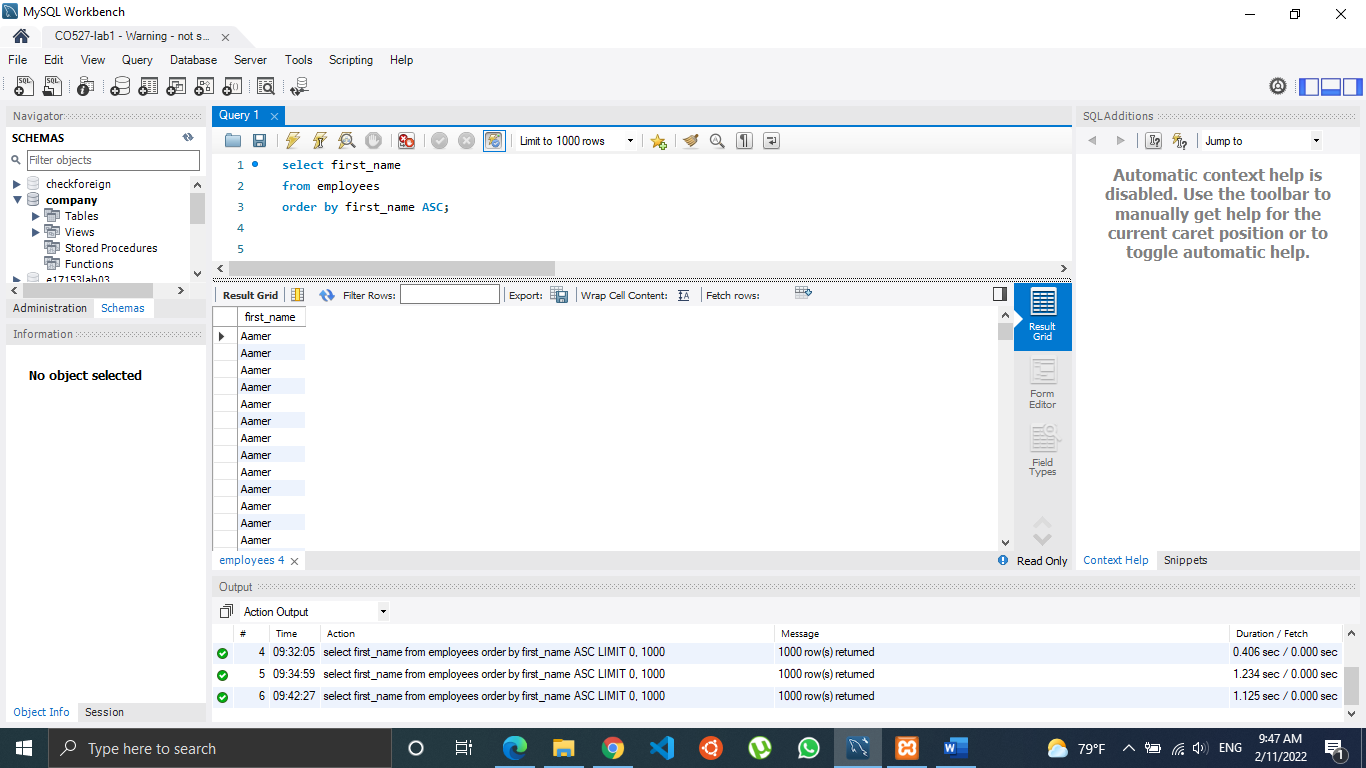
**CO527- Advanced Database Systems**

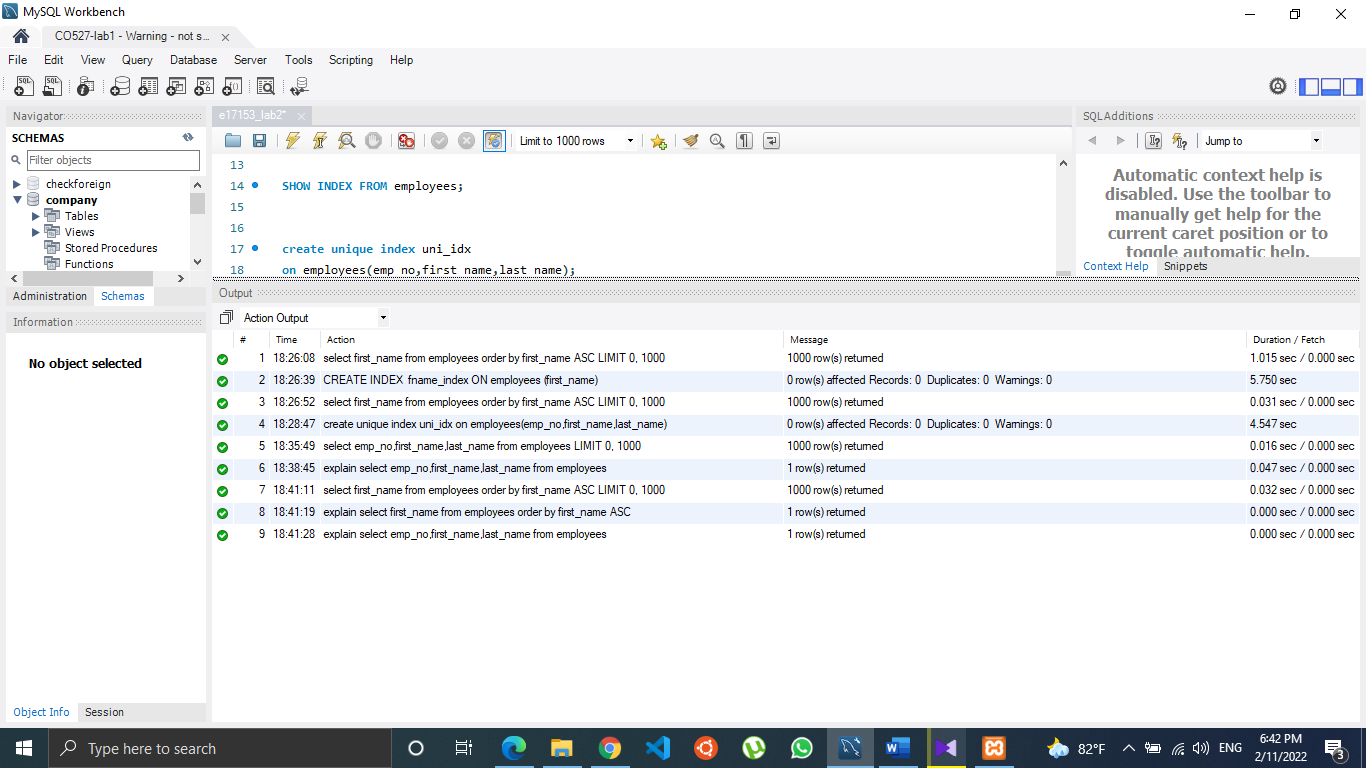
**Lab 02- Indexing**

**Karunachandra R.H.I.O.**

**E/17/153**

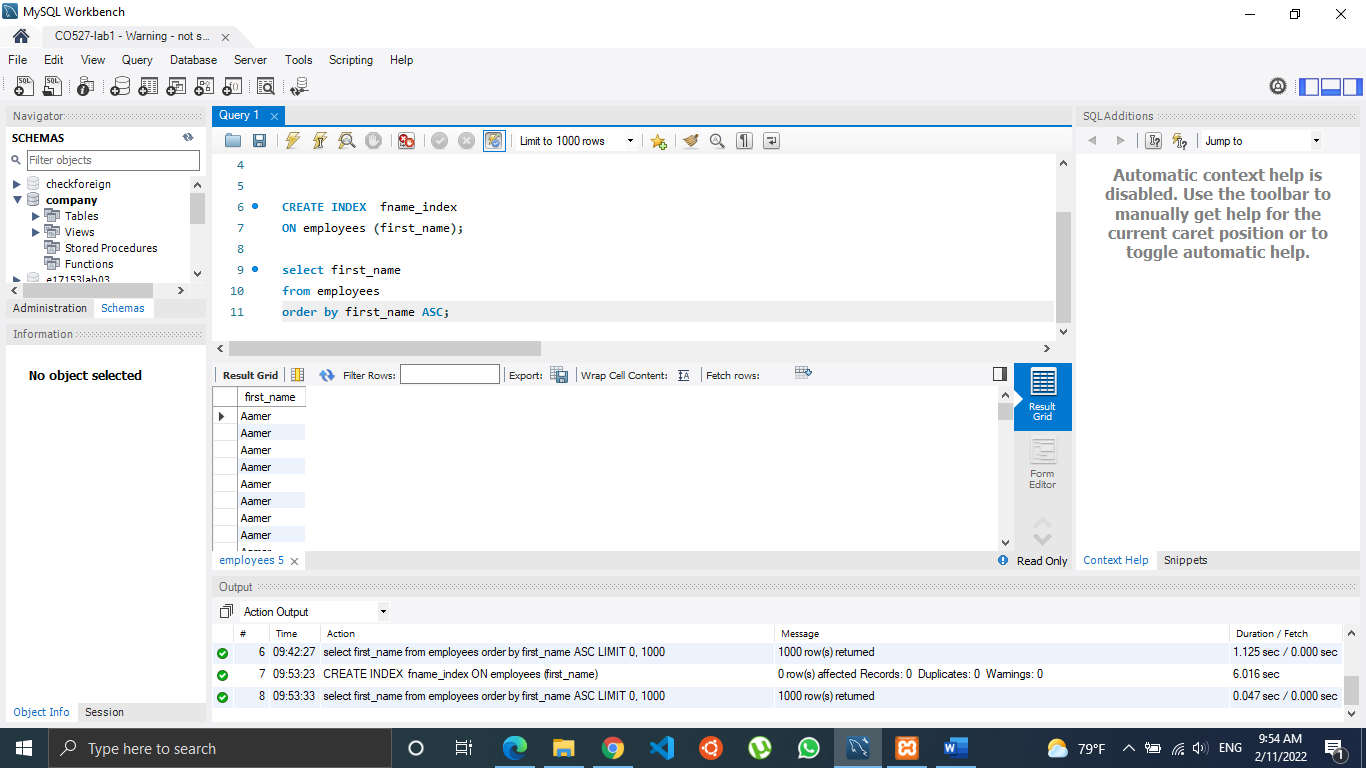
1)Assuming no indexes are used, record the query execution time for retrieving all the employees by first name in ascending order.

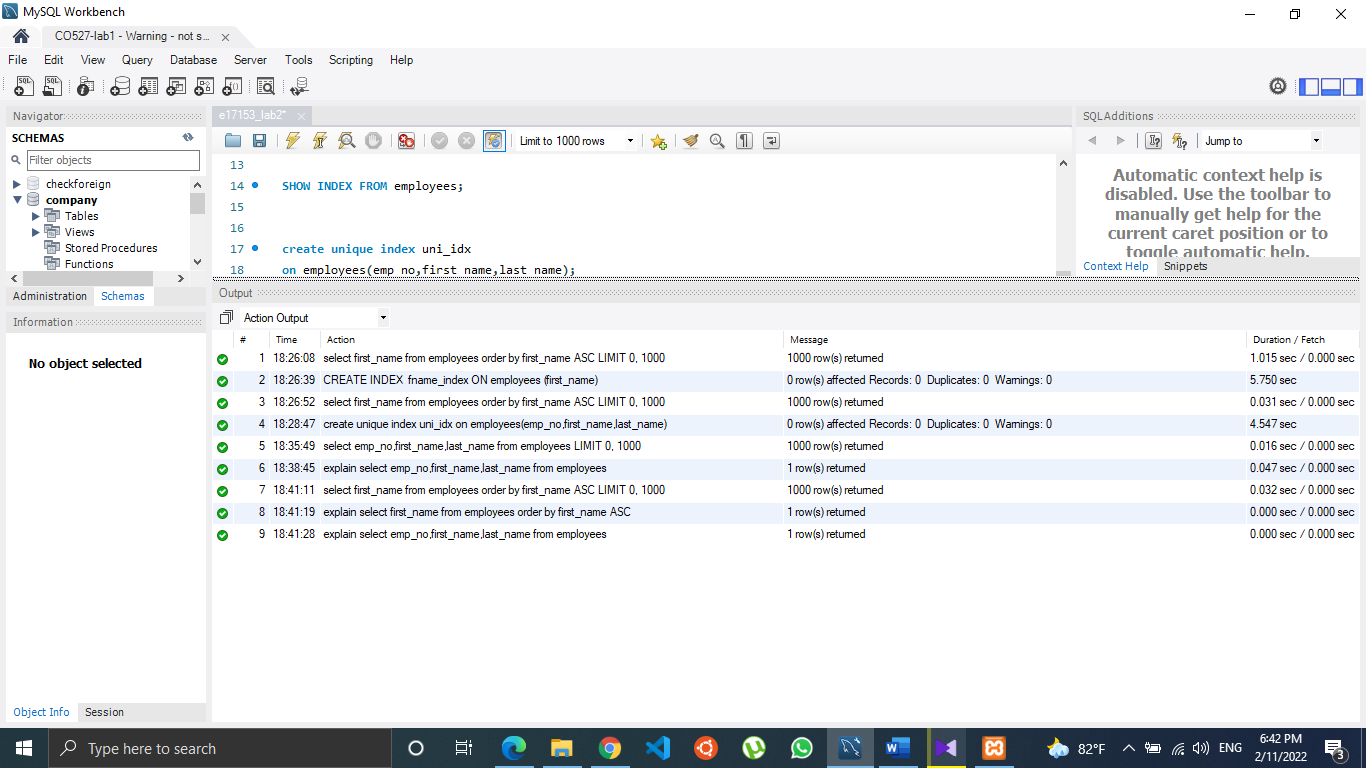




Query execution time was **1.015s**

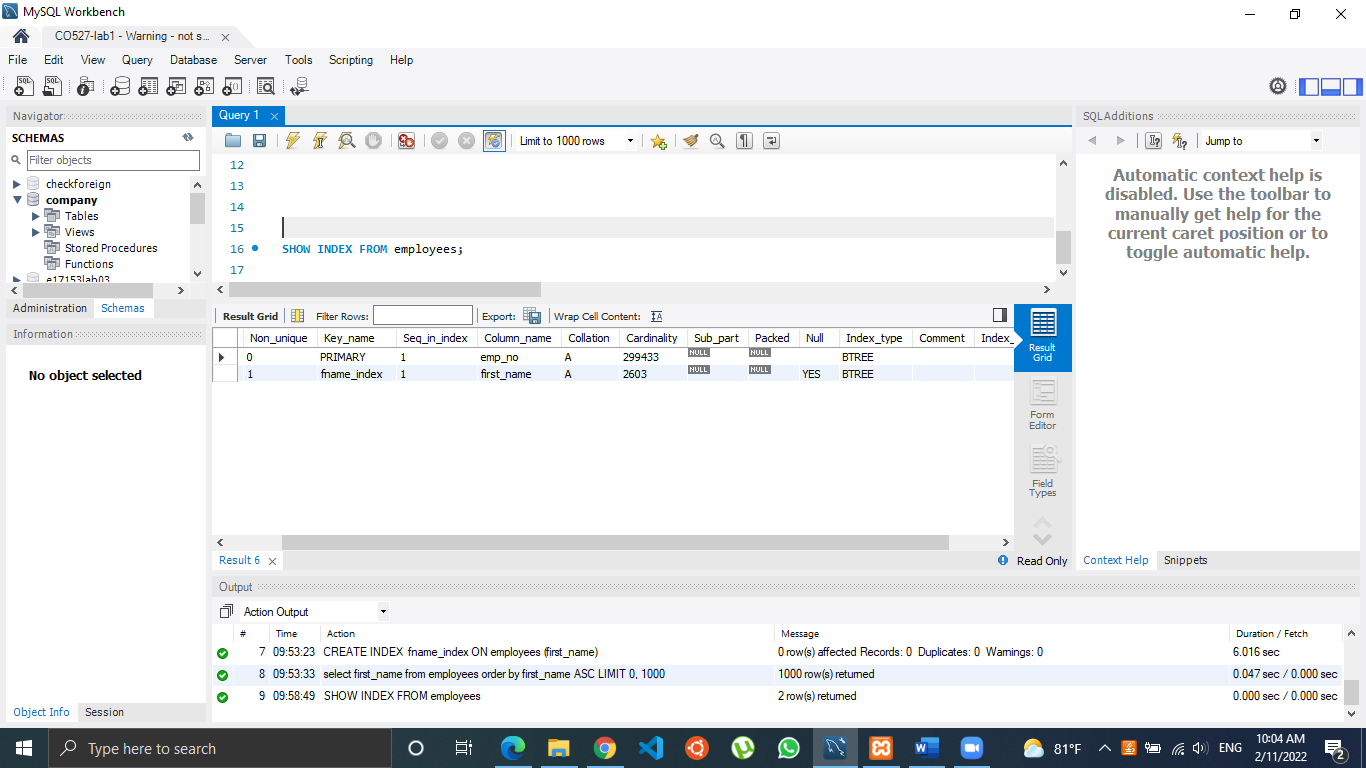
2)Create an index called fname\_index on the first\_name of the employee table. Retrieve all the employees by first name and record the query execution time. Observe the performance improvement gained when accessing with index.





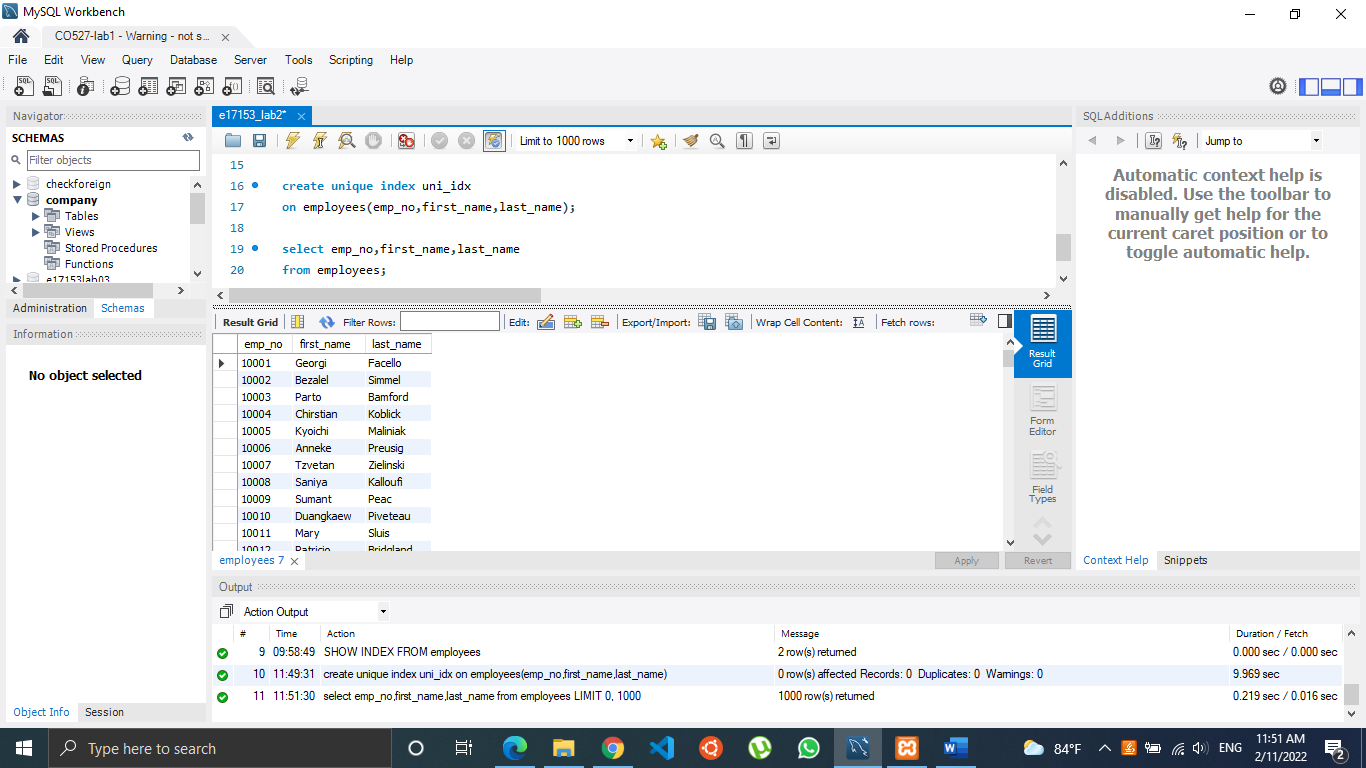
Query execution time was **0.031s**

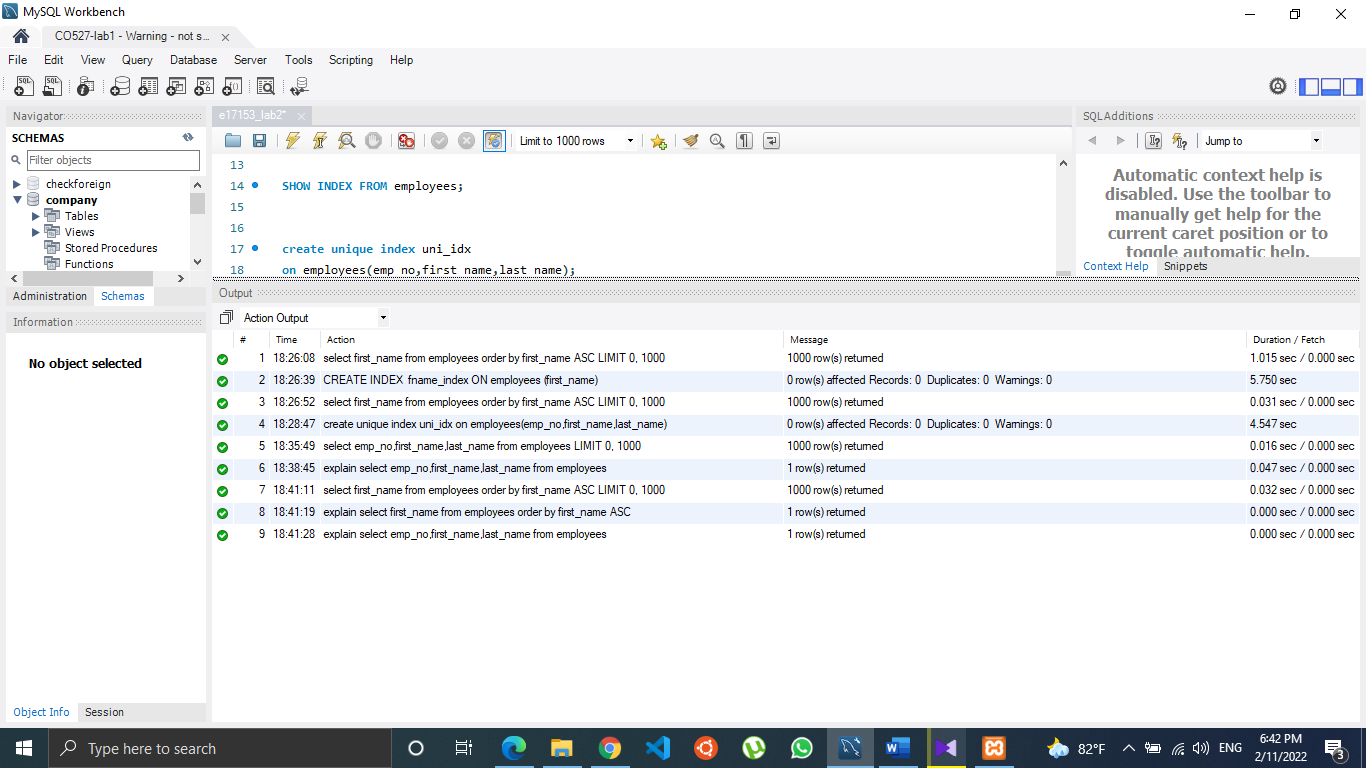
3) Which indexing technique has been used when creating the above index? Hint: You can use SHOW INDEX FROM [mytable]; to see details of your indexes.



**BTREE (Binary Tree)** is the technique used in creating indexes.

4) Create a unique index on emp\_no, first\_name and last\_name of employees table. Retrieve all the employees by emp\_no, first\_name and last\_name. Observe if there is any performance improvement with respect to question1. If not, explain any possible reason.





Query execution time was **0.016s**

Part 1 execution time is higher than part 4 execution time because, in the part 1 we didn’t use indexing. As it is shown above part 1 time is greater than part 2 time where we used the same query but with indexing. When comparing with part 4, the queries are bit different because in the part 4 we retrieve three fields. Even though, part 4’s binary search algorithm will speed up the execution than part 1’s normal search algorithm. So, it took less time in part 4.

This time difference can be affected by other programs running in the background too.

5) Take the following 3 queries

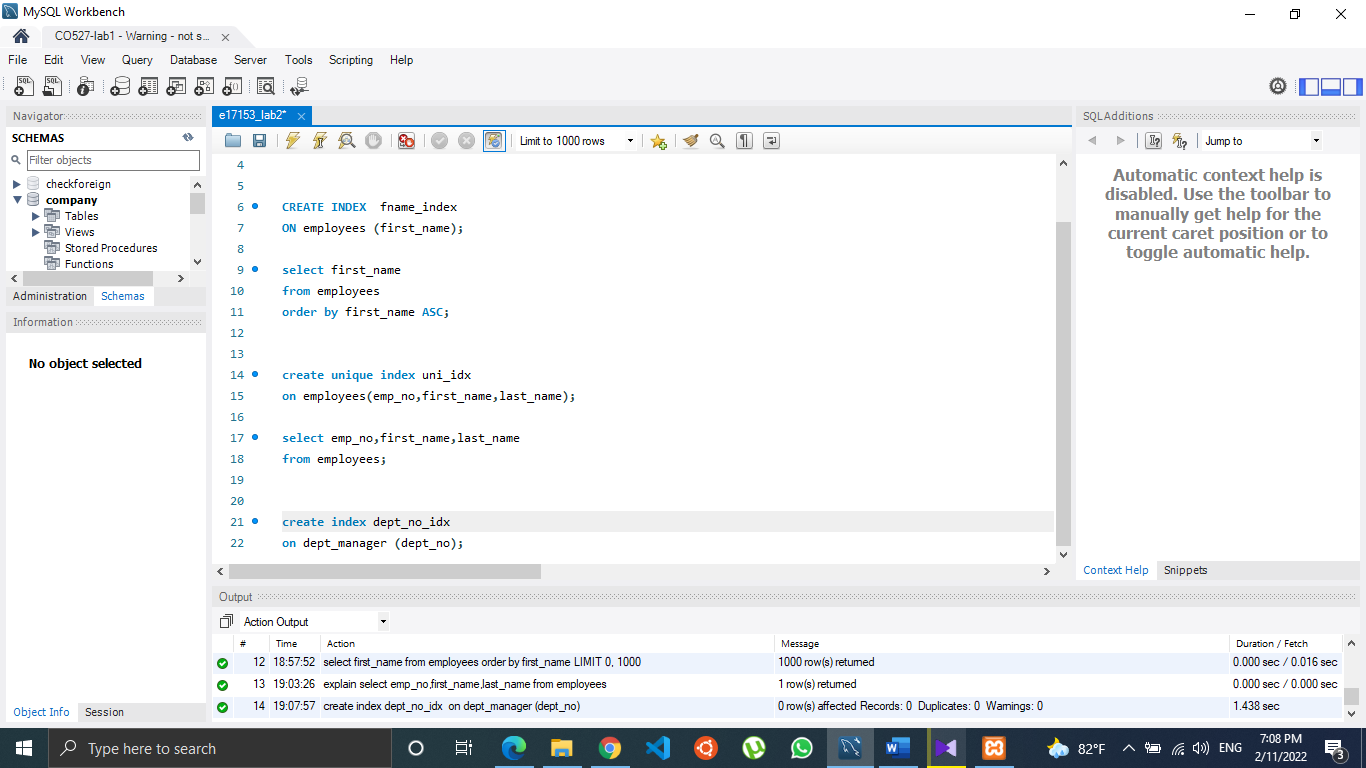
A. select distinct emp\_no from dept\_manager where from\_date>=’1985-01-01’ and dept\_no>= ’d005’;

B. select distinct emp\_no from dept\_manager where from\_date>=’1996-01-03’ and dept\_no>= ’d005’;

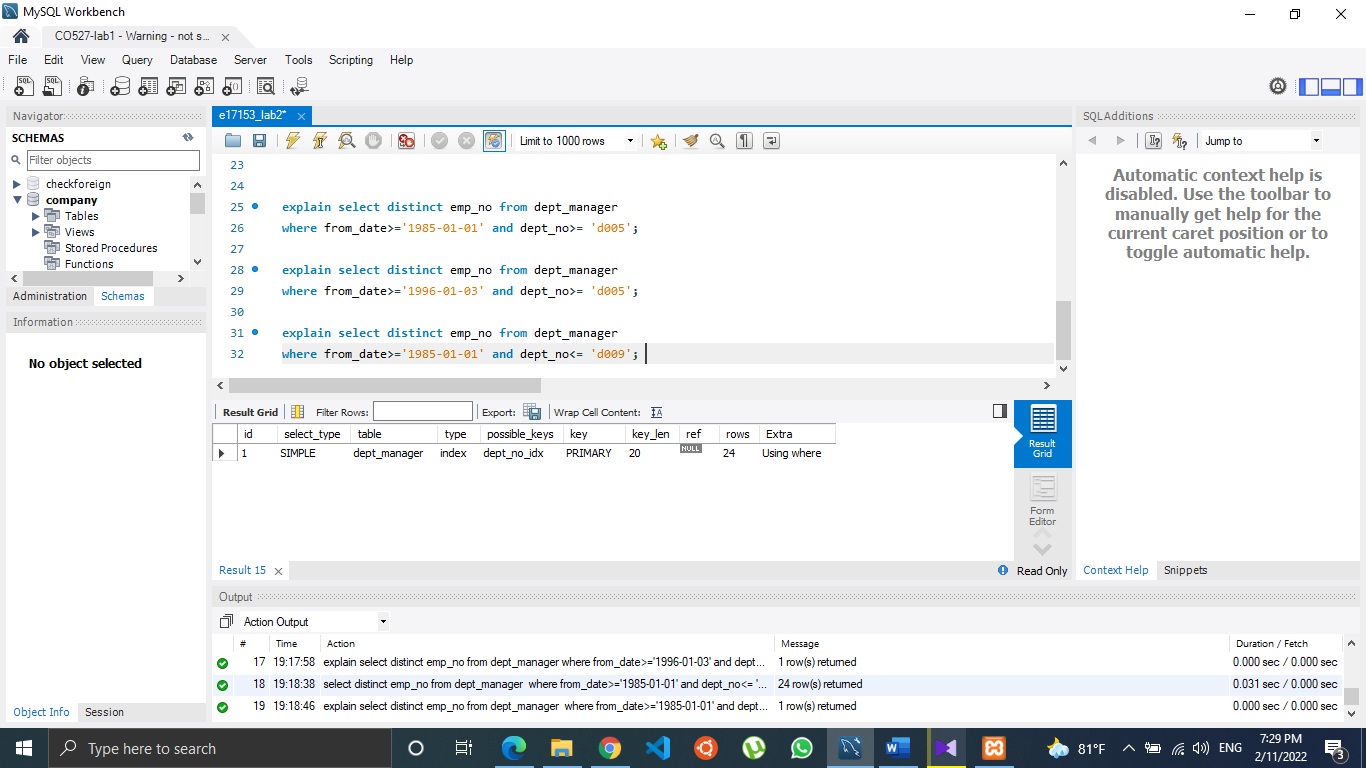
C. select distinct emp\_no from dept\_manager where from\_date>=’1985-01-01’ and dept\_no<= ’d009’;

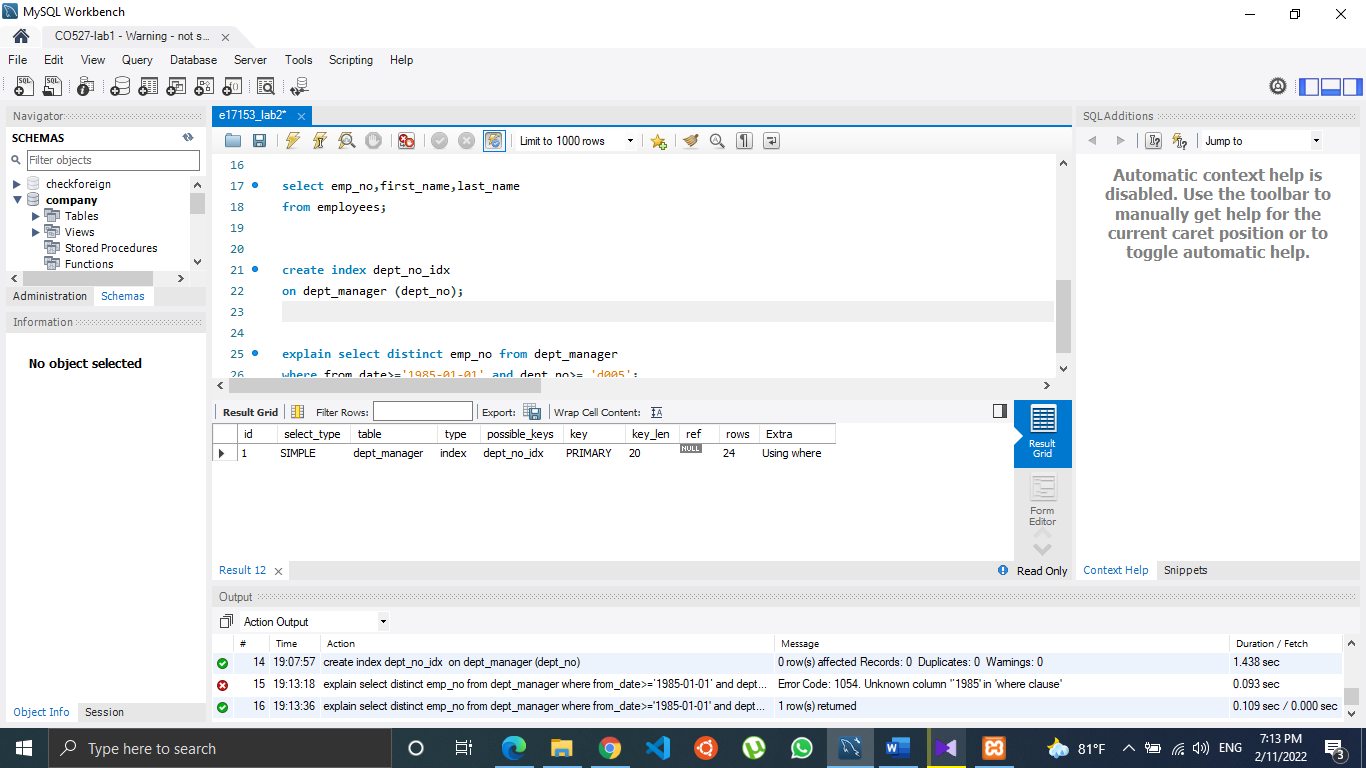
I) Choose one single simple index(i.e index on one attribute) that is most likely to speed up all 3 queries giving reasons for your selection.

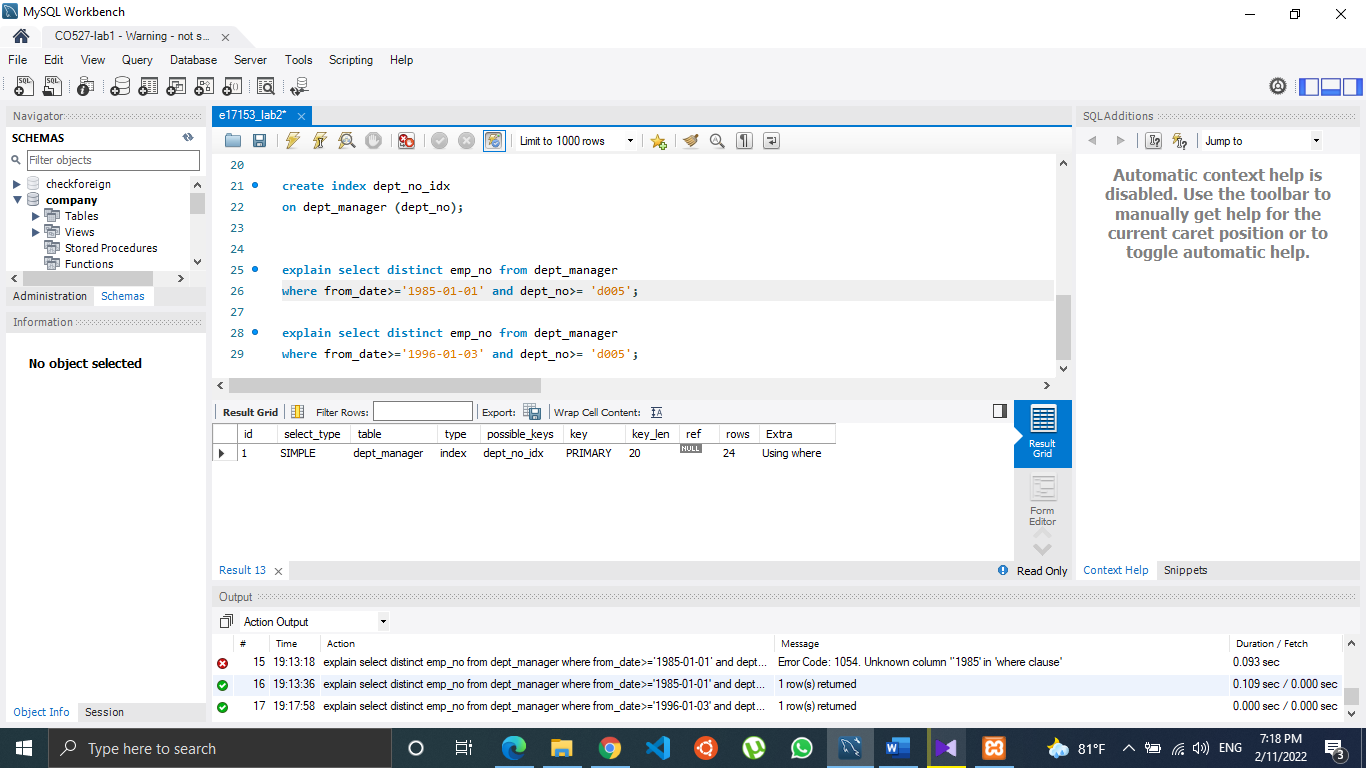
dept\_no - Usually indexes work well with the attributes that are used under the WHERE clause. Here both from\_date and dept\_no are under the WHERE clause but adding a unique constraint to an index further increases the write speed. Also, arranging from dept\_no makes more sense than arranging from from\_date. So, dept\_name will be the most suitable choice.

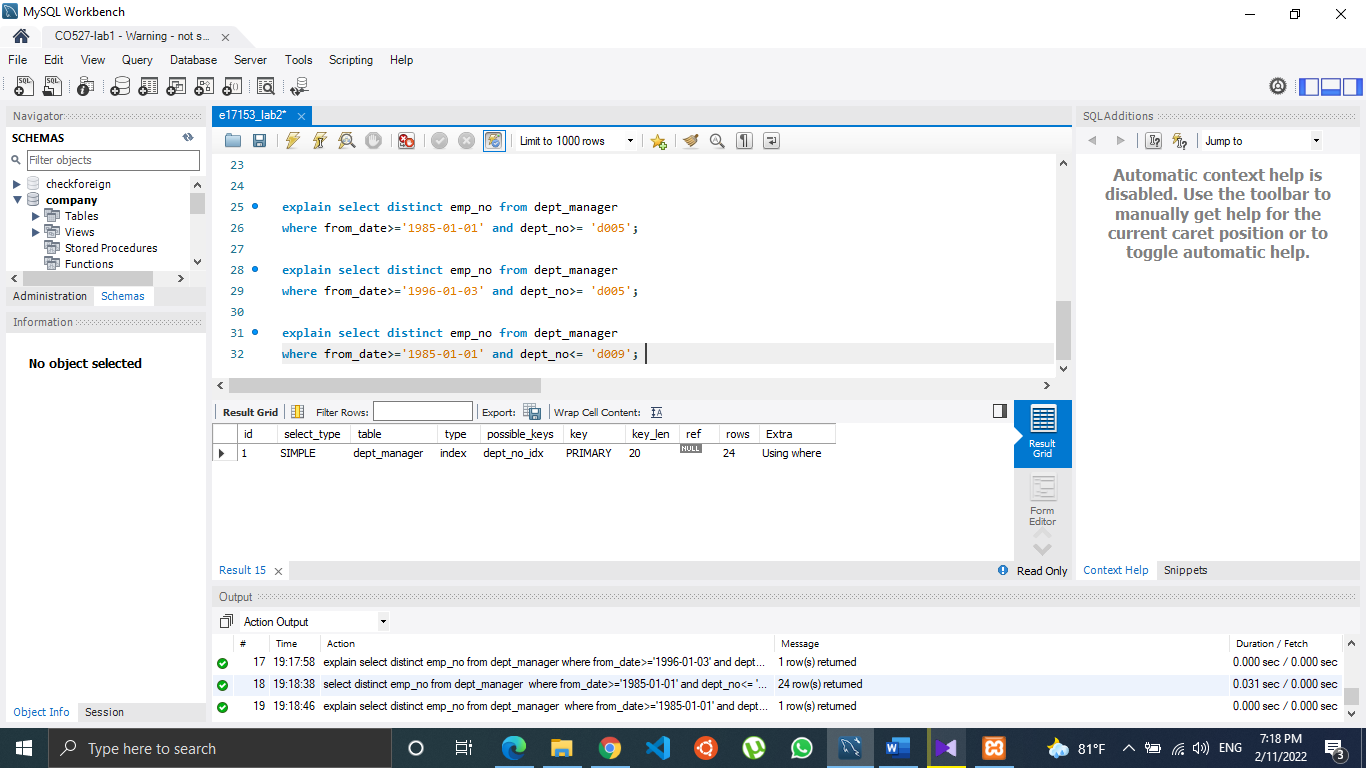


II) For each of the 3 queries, check if MySQL storage engine used that index. If not, give a short explanation why not. You can prefix your select queries with EXPLAIN EXTENDED or with EXPLAIN to display a query execution plan. (Note that in MySQL InnoDB engine uses a clustered index usually on the primary key of the table, by default. We only care about the index you create.









Here in all three cases, the index we created are in the ‘possible keys’ column. But PRIMARY KEY is used as the index.

Usually in mySQL InnoDB, a special index called **“clustered index”** is used and in the most cases it is the primary key of the particular table. Indexes other than the clustered index are called **“secondary indexes”.** Here,’dept\_no\_idx’ is the secondary index. When there is a selection between indexes, InnoDB uses the index that finds the smallest number of rows. (The most selective index) Since the primary key constraint cannot be NULL, it will perform a boost in the occurrence when going through the tables. Hence, PRIMARY KEY is used as the index in all three cases instead of the secondary index we created.

6) Consider the queries you wrote for questions 2 - 10 in Lab 01 assignment. Give with short explanations, which attributes on which relations should be used for creating indexes that could speed up your queries.

|  |  |  |
| --- | --- | --- |
| Query no | Index | Explanation |
| 2 | last\_name in employees | In the query, ‘last\_name’ field is selected,counted and grouped data by it. |
| 3 | title in titles | In the where clause, we compare the title as “Engineer”. So, it would speed up the execution if title was indexed. |
| 4 | title in Titles and sex in employees | In the where clause, we compare the title as “Senior engineer” and sex as “F”. So, it would speed up the execution if they were indexed. |
| 5 | salary in salaries | Here in the where clause we search for employees who  have salary greater than 15000, creating index  will sort the salary column and it will speed up the query. |
| 6 | birth\_date and hire\_date in  employees | In the where clause, we compare age and hire date. To calculate age, birth date is used. So, if those two columns are sorted, it will speed up the execution. |
| 7 | dept\_name in departments | In where clause it is compared that id dept\_name is equal to “Human Resources”. If that column is indexed, it will reduce the execution time. |
| 8 | salary in salaries and  dept\_name in departments | Here we execute this query based on salaries of finance department employees. So, these two indexes will speed up the query. |
| 9 | salary in salaries | In where clause “salary” column is used to find the avg and for the select operation. |
| 10 | salary in salaries and title in titles | Here we use salary to find some averages and title is checked in one place. So, indexing those two will increase the efficiency of the query. |

7) Assume that most of the queries on a relation are insert/update/delete. What will happen to the query execution time if that relation has an index created?

* When there is an **INSERT**, it means new data is writing to the tables. If the relation has an index created, the query execution will have to write those data into tables and also into the indexes. That will consume more time.
* **DELETE** can be affected in both ways, speeding time and slowing time. If we have indexes for a deleting query, it will make it faster to find the particular data elements. At the same time, it will slow the process because now we have to delete them from both tables and indexes. So, the optimal way will be if there is a small amount of data deletion from a large data set.
* **UPDATE** means delete some data elements and insert another element there. So, the same principals will apply according to the case.